ESPECIFICACIÓN E IMPLEMENTACIÓN DE UNA LIBRERIA PARA EL ESTUDIO DE LA SECUENCIA DE EXCENTRICIDADES DE UN GRAFO

por Fernando Gil

21 de septiembre de 2008

0.1. Agradecimientos

La realización de este trabajo final de carrera se ha debido a mucha gente.

Principalmente a mi profesor de proyecto Joan Gimbert cuyo apoyo e interés ha sido constante y a otros tantos profesores que durante estos años de carrera me han enseñado a "usar el coco".

También a mis padres que por fín verán a su hijo acabar la carrera, fuente de la mayoría de sus preocupaciones (que no son pocas).

A mi otra parte del ser, Rachel, por todo lo que tú ya sabes y por todo lo demás. Además del excelente apoyo bibliográfico que me has dado.

A mis compañeros de universidad y sin embargo amigos, ¡gracias por estar ahí! Una mención especial para mi colega Joan, sin él mis grafos serían más feos y pixelados.

A mis amigos que han tenido que aguantar mis neuras exclusivamente.

Y para Mari y Raúl por hacerme acordar que tenía que comer para vivir...

Finalmente a mi compañero de trabajo, Antonio Belenguer.

¡Gracias!

Índice general

Índice de figuras

List of Algorithms

Índice de cuadros

Capítulo 1

Presentación

1.1. Preliminares

El proyecto se encargará básicamente de dos dinámicas iniciales:

1 El estudio del sistema Nauty creado por Brendan D. McKay

2 La creación de una herramienta para calcular las excentricidades de un grafo.

1.2. Objetivos a conseguir

▷ Enumerar todas las secuencias de excentricidades de un grafo de orden pequeño.

▷ Realizar un catálogo de las secuencias de excentricidades de los grafos de órdenes pequeños.

▷ Generar aleatoriamente ciertas clases de grafos, y determinar su secuencia de excentricidades.

▷ Estudiar el comportamiento de la secuencia de excentricidades de un grafo ante diferentes transformaciones elementales del mismo.

1.3. Estructura de la memoria

▷ **CAPÍTULO 1: INTRODUCCIÓN**
Presentación del trabajo y sus objetivos a asumir.

▷ **CAPÍTULO 2: TEORIA DE GRAFOS**
Teoría básica para el entendimiento de capítulos posteriores.

▷ **CAPÍTULO 3: REPRESENTACIÓN INFORMÁTICA DE GRAFOS**
Manera de representar grafos informáticamente.

▷ **CAPÍTULO 4: NAUTY**
Utilidades referidas de este programa para el trabajo.

▷ **CAPÍTULO 5: LA HERRAMIENTA**
Código y utilidades de la herramienta.

▷ **CAPÍTULO 6: CONCLUSIONES Y TRABAJOS FUTUROS**
Conclusión del proyecto y posibles continuaciones del mismo.

1.4. Lecturas recomendadas

Para un mejor entendimiento del trabajo en la parte final de esta memoria se puede encontrar un listado bibliográfico.

▷ **Parte Teórica:** Conocimientos teóricos básicos sobre la Teoría de Grafos. *Ver* [10], [5], [7], [13], [6], [14], [15], [1], [12], [3]

▷ **Parte Nauty:** El programa de diseño de Brendan D. McKay. *Ver* [11]

▷ **Parte Programación:** Ayuda para programación en entorno JAVA, para manejo del Látex y algoritmos de programación. *Ver* [9], [4], [8]

▷ **Parte Futura:** Dinámicas y teorías para trabajos futuros. *Ver* [2]

Como veremos durante el trabajo muchos de estos han sido citados especialmente en algunas partes.

Parte I

Teoría de grafos

Capítulo 2

Conceptos básicos de la Teoría de Grafos

2.1. La Teoría de Grafos

La teoría de grafos, que nació como una rama de la Topología, se ha convertido hoy en día en una herramienta matemática indispensable en campos tan diversos como la investigación operativa, la lingüística, la química, la física, la genética, la teoría de redes o la teoría de la decisión.

Es por ello, que para casi cualquier rama de la ciencia, se hace necesario el conocimiento de las ideas básicas que sustentan a la denominada teoría de grafos.

Este capítulo tiene por objeto introducir al lector en los conocimientos básicos sobre grafos de manera que podamos explicar más adelante los apartados más complejos del trabajo.

Se recomienda diferentes documentos (*Ver* [10], [5], [7], [13], [6], [14], [15], [1], [12]), de forma que el lector encuentre más facilidades en el entendimiento de este trabajo.

2.1.1. Ejemplos históricos

A continuación se exponen dos ejemplos de la utilización de grafos en problemas de la vida cotidiana.

El primero es uno de los más famosos problemas con los que nació la teoría de grafos y el segundo contiene un problema de combinatoria. Son dos ejemplos clásicos de la dinámica diferente que puede tomar la extensa teoría de grafos.

2.1.1.1. Los puentes de Königsberg

La teoría de grafos nació, en parte, en el siglo XVIII en la ciudad de Königsberg (Prusia) que tenía dos islas y siete puentes según indica la figura:

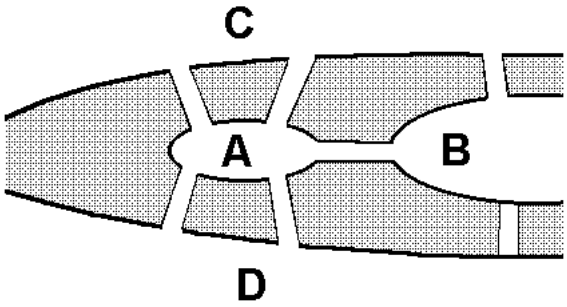

Figura 2.1: Diagrama de la ciudad de Königsberg

El alcalde de la ciudad escribió a Euler planteándole la siguiente cuestión:

-¿Es posible que una persona cruce los siete puentes pasando por cada uno de ellos una sola vez?

Euler probó que era imposible lo que el alcalde proponía. Sustituyó la isla y las orillas por puntos y los puentes por líneas que unían dichos puntos, obteniendo así el siguiente grafo:

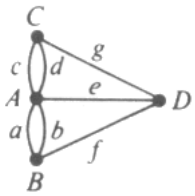

Figura 2.2: Grafo de Königsberg

El grafo será *recorrible* si existe un camino que contenga todos los vértices y que pase por cada arista exactamente una vez.

Supongamos que haya un camino que no empiece ni termine en un vértice u. Cada vez que el camino llegue a u debe de salir por otro que no haya sido utilizado. De esa manera, las aristas del camino incidentes en u deben aparecer de dos en dos, es decir, el grado de u es *par*. Así, si el grado de v es *impar*, el camino debe empezar y terminar en otro vértice impar.

Basándose en ese razonamiento, se deduce, a lo largo del camino, no puede haber más de dos vértices que sean impares. El grafo anterior tiene sus cuatro vértices impares, por lo que no es recorrible.

2.1.1.2. El Teorema Matrimonial de Hall

Otro problema solucionable mediante grafos es el causante de este teorema.

Dado un conjunto m de muchachos, cada uno de los cuales conoce a varias chicas. ¿En qué condiciones se pueden formar los matrimonios de tal forma que cada uno de los muchachos se case con una chica que conoce?

La solución que propuso Hall fue la siguiente:

Una condición necesaria y suficiente para la solución del problema matrimonial es que cada conjunto de α jóvenes conozca colectivamente α chicas al menos ($1 \leq \alpha \leq m$), para cada valor de α, donde m es el total de chicos.

2.2. Conceptos básicos sobre grafos y digrafos

2.2.1. Conceptos sobre grafos

En este apartado pasaremos a documentar las definiciones previas matemáticas precisas para asumir la teoría necesaria, con especial incapié en la *Definición 5* que explica la *excentricidad*.

(Recomendada la lectura de [3] *)*

Definición 1:

Un *grafo* $G = (V, E)$ es una estructura combinatoria constituida por un conjunto finito $V = V(G)$ de elementos denominados *vértices* y un conjunto $E = E(G)$ de pares no ordenados de vértices distintos denominadas *aristas*. Si la arista $e = \{ u, v \}$ relaciona los vértices u y v se dice que u y v son vértices *adyacentes* y también que el vértice u (o v) y la arista e son *incidentes*.

El número de vértices de G, $|V(G)|$, es el órden del grafo y el número de aristas $| E(G) |$ es su medida.

Definición 2:

El grado de un vértice v en un grafo $G = (V, E)$, que denotaremos por $g(v)$, es el número de aristas de G incidentes con v.

Definición 3:

Dado un grafo $G = (V, E)$, una secuencia de vértices $u_0, u_1, \dots, u_n$ con $u_{i-1}u_i \in E$, $1 \leq i \leq n$, se denomina un *recorrido* R de longitud n entre u_0 y u_n.

Definición 4:

Si entre todo par de vértices u y v de G existe un recorrido, el grafo se denomina *conexo* y, en este caso, la *distancia* entre u y v, $d(u, v)$, es la longitud mínima de los recorridos entre esos vértices.

Propiedades de interés de las distancias:

- $d(u, v) \geq 0$ y $d(u, v) = 0$ si y solo si $u = v$

- $d(\ u,\ v\ ) = d(\ v,\ u\ )$

- $d(\ u,\ v\ ) \leq d(\ u,\ w\ ) + d(\ w,\ v\ )$

En el caso de que no haya ningun recorrido entre dos vértices u y v, se define $d(\ u,\ v\ ) = \infty$.

Definición 5:

La *excentricidad* de un vértice $u \in V(\ G\ )$ es la máxima de las distancias entre u y el resto de vértices de G.

Podemos ver un ejemplo a continuación:

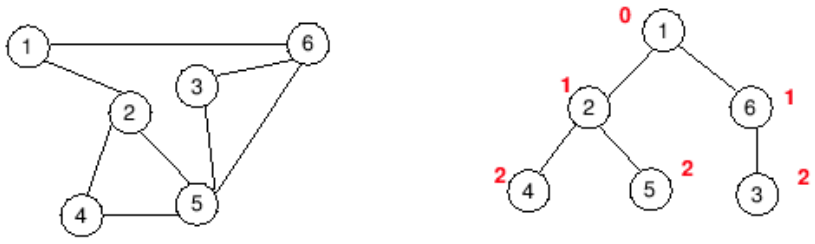

Figura 2.3: Excentricidad de un vértice

En este caso estamos calculando la excentricidad del **vértice *1*** que es igual a *2*.

Definición 6:

El diámetro de G es el máximo de las excentricidades de todos sus vértices.

Propiedad:

- Si G tiene orden n, entonces $diam(G) \leq n - 1$

Definición 7:

El radio de G es el mínimo de las excentricidades de todos sus vértices.

Un vértice de un grafo es **central** si la máxima distancia que lo separa de otro vértice es el radio.

Propiedad:

- $rad(G) \leq diam(G) \leq 2rad(G)$

Esta relación dejará de cumplirse en los digrafos como veremos a continuación

2.2.2. Conceptos sobre digrafos

(Para este apartado mirar el documento [5])

Definición 8:

El concepto de grafo dirigido, o *digrafo*, deriva directamente del concepto de grafo, exigiendo que las aristas, ahora denominadas *arcos*, sean pares ordenados de vértices distintos. De esta manera, un digrafo $G = (V, E)$ es una estructura combinatoria formada por el par (V, E) de conjuntos finitos tales que E es un conjunto de pares ordenados de elementos distintos de V. Si $a = (u, v) \in E$ decimos que el vértice u es adyacente hacia el vértice v y que v es adyacente desde u.

Definición 9:

Dados dos vértices u y v, un recorrido de longitud $h \geq 0$ desde u hasta v es una secuencia de vértices $u = u_0, u_1, \ldots , u_n = v$ con $u_{i-1}u_i \in E(G), 1 \leq i \leq n$.

Definición 10:

Si entre todo par de vértices u y v de G existe un recorrido de u hasta v, el digrafo se denomina *fuertemente conexo* y, en este caso, la *distancia* de u a v, $d(u, v)$, es la longitud mínima de los recorridos de u a v. En un digrafo conexo G, la distancia no es simétrica, a diferencia de lo visto en los grafos. Hay casos en los que $d(u, v)$ y $d(v, u)$ no tienen porque coincidir.

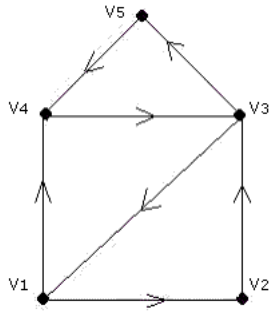

Figura 2.4: Digrafo

- Observar que *diam(G)* $\nleq$ *2rad(G)*

Para comprobar esta propiedad disponemos del siguiente digrafo:

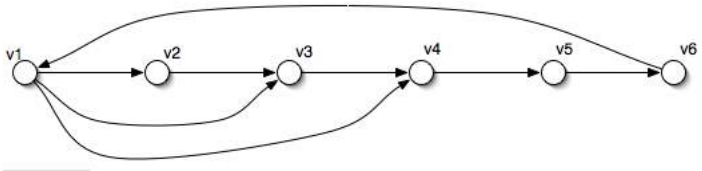

Figura 2.5: Digrafo para comprobación de la propiedad

Vemos que excentricidad tiene cada vértice:

▷ **v1** $\rightarrow 3$

▷ **v2** $\rightarrow 5$

▷ **v3** $\rightarrow 5$

▷ **v4** $\rightarrow 4$

▷ **v5** $\rightarrow 3$

▷ **v6** $\rightarrow 3$

En este caso nos encontramos con un diámetro de 5 y un radio de 2 por lo que *diam(G)* $\nleq$ *2rad(G)* se cumple.

La *distancia* cumplirá la propiedad de *simetria* (para cualquier par de vértices) si el digrafo es *simétrico* (en cuyo caso se identificaria como grafo), es decir, siempre que tengamos el arco *(u, v)* tendremos el arco en sentido opuesto *(v, u)*.

2.3. Secuencias de excentricidades

El estudio de las propiedades de las distancias de los grafos y digrafos es un área clásica dentro de la teoría de grafos.

Las aplicaciones reales de los cálculos de excentricidades tienen que ver con minimizar las distancias entre puntos. Es uno de los factores principales a la hora de situar un determinado recurso (hospitales, aereopuertos, correos, etc). Estos recursos deberían estar situados lo más céntrico posible, según los grafos en los vértices centrales y he aquí donde radica el problema de la localización de recursos. Esta minimización se puede obtener empleando propiedades de las distancias de grafos y digrafos como son las excentricidades.

En el anterior apartado hemos definido la excentricidad *(Definición 5)*, a partir de ahora ahondaremos más en este concepto de la teoría de grafos.

Definición 11:

Una secuencia no decreciente *S:* a_1, ... , a_n de enteros no negativos se denominará *secuencia de excentricidades* si existe un grafo G de orden n cuyos vértices v_1, ... , v_n cumplen que $e(v_i) = a_i$ para toda i. En este caso diremos que S es la *secuencia de excentricidades* de G.

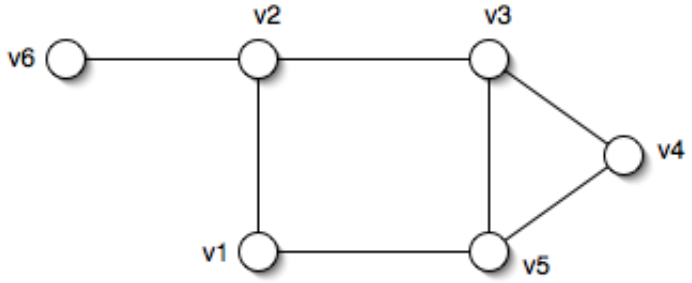

Figura 2.6: Grafo y su secuencia de excentricidades

El grafo de la figura es de orden 6 y sus vértices son v1, v2, v3, v4, v5 y v6.
Pasamos a calcular la excentricidad de cada vértice:
e(v1)= 2, e(v2)= 2, e(v3)= 2, e(v4)= 3, e(v5)= 3, e(v6)= 3
Así pues, la secuencia de excentricidades del grafo es ***2,2,2,3,3,3***
Esta definición es igualmente válida para el caso de los digrafos.
(Veáse [10] *)*

Propiedades:

Si *S:* a_1, ... , a_n es la secuencia de excentricidades de un grafo G dada en orden no decreciente entonces:

- $a_1 \leq n/2$
- Si k es un entero tal que $a_1 < k \leq a_n$, entonces $a_i = a_{i+1} = k$ para al menos un valor de i *(2* $\leq i \leq$ *n - 1)*
- $a_n \leq min$ *{ n - 1,* $2a_1$*}*

Estas propiedades no se cumplen para la parte de digrafos. Veamos a continuación cuales de ellas se satisfacen.

Propiedades:

Si *S:* a_1, ... , a_n es la secuencia de excentricidades de un digrafo G entonces:

- $a_{i+1} \leq a_i + 1$ para todo i *(1* $\leq i \leq$ *n - 1)*
- $a_n \leq n - 1$

Ahora bien, las secuencias de excentricidades cumplen diferentes propiedades tanto para grafos como para digrafos.

Propiedad:

Una secuencia *S:* a_1, ... , a_n con m distintos valores es una secuencia de excentricidades si y solo si alguna subsecuencia de S, con m distintos valores es una secuencia de excentricidades.

Propiedad:

Una secuencia constante S de longitud p ≥ 2 , *S: a, ... , a,* con a $\in Z^+$, es una secuencia de excentricidades si y solo si $a \leq p - 1$

2.3.1. Mecanismo empleado para el cálculo de excentricidades

Se hace necesario pues, el cáculo de las excentricidades de cada vértice de un grafo o digrafo y, por tanto, el conocimiento del cálculo de distancias.

Mediante el algoritmo BFS podemos resolver este problema. Este algoritmo se aplica a cada vértice del grafo o digrafo, obteniendo así una matriz de distancias.

Este algoritmo se basa en dos estructuras básicas, una lista de vértices pendientes de entrar y una cola de vértices que ya han entrado.

Algorithm 1 Algoritmo BFS

1. *distancia*[u]←0
2. l_{aux}←V - {u}
3. c ←{u}
4. **mientras** (¬*vacia*(c))
5. u ←*primero*(c)
6. **para cada** ((w ∈l_{aux}) **y** (w adyacente a v))
7. *distancia*[w]←*distancia*[v] + 1
8. c←*insertar*(w)
9. l_{aux}← l_{aux} - {w}
10. **fin para cada**
11. c ←*eliminar*()
12. **fin mientras**

A partir de la matriz de adyacencia del grafo y de estas dos estructuras auxiliares iremos agregando para cada vértice todos sus vértices adyacentes, que se encuentran en la lista de vértices pendientes, eliminándolos a continuación de esa lista.

De esta manera, mientras queden elementos en la cola, se irán comprobando todas las distancias entre un vértice y el resto de vértices del grafo o digrafo. *(Véase [9] para Algoritmo)*

Las funciones *insertar* y *eliminar* corresponden a la estructura de cola.

2.3.2. Secuencias de excentricidades minimales

Dicho esto y viendo la definición de secuencia de excentricidades deberiamos ser capaces de hallar las **secuencias de excentricidades minimales**.

Las secuencias de excentricidades minimales son aquellas que no contienen ninguna otra secuencia de excentricidad.

Estas secuencias de excentricidades son las más básicas que se pueden investigar a partir de los grafos de órdenes definidos de tal manera que muchas de las siguientes secuencias de excentricidades serán combinaciones de las minimales.

La notación que utilizaremos para dichas secuencias estará ordenada de menor a mayor y los números estarán elevados a otros que serán las veces que se repita dicha excentricidad en nuestro grafo.

Relación de todas las secuencias de excentricidades de grafos conexos de orden $\leq$4:

Para orden 2:

1^2

Para orden 3:

1^3, $1^1 2^2$

Para orden 4:

1^4, $1^1 2^3$, $1^2 2^2$, 2^4, $2^2 3^2$

De aquí las **secuencias minimales** son:

1^2, $1^1 2^2$, 2^4, $2^2 3^2$

Ejemplo:

Quisieramos probar que 1^3 *es una secuencia minimal.*

Si 1^3 fuera minimal no podría contener ninguna otra secuencia minimal que pudiera formarla.

Ponemos 1^3 de otra manera como 1, 1, 1 y comprobamos si existe un minimal escondido en dicha secuencia. Observamos claramente que 1, 1 es 1^2, el minimal extraido para grafos de orden 2 y que está contenido en 1^3 por lo que dicha secuencia no es minimal.

Según los grafos vayan aumentando su orden se vuelve más complicado solucionar el problema de las listas de minimales. Nuestra herramienta intentará solucionar la labor en este campo como veremos en la parte III.

Capítulo 3

Representación de grafos y digrafos

Tanto para la representación de un grafo o un digrafo como un par *(V,E)* de conjuntos finitos, donde los elementos de E tienen la forma *{u, v}* o *(u, v)*, siendo u y v vértices diferentes de V, como la representación mediante un dibujo, no resultan adecuadas si queremos que sean procesadas por un ordenador.

A continuación presentamos las diferentes representaciones.

3.1. Representación matricial

3.1.1. Matriz de adyacencias

Definiremos la *matriz de adyacencia* de un grafo $G = (V,E)$ de orden n, con el conjunto de vértices $V=\{v_1,v_2,...,v_n\}$, como la matriz cuadrada $A(G) = (a_{ij})$, siendo a_{ij} los elementos de la matriz de orden n x n.

$$a_{ij} = \begin{cases} 1 \\ 0 \end{cases}$$

1 ⇒Si v_i y v_j son adyacentes.

0 ⇒En otro caso.

La matriz de adyacencia es simétrica con elementos nulos en la diagonal. De

otra manera el número de elementos iguales a 1 en la fila (o columna) i de *A(G)* es *g(v_i),* el grado del vértice v_i o, lo que es equivalente, el número de caminos de longitud 1 que comienzan en el vértice v_i.

La matriz de adyacencia de un digrafo no es, en general, simétrica, no obstante, preserva los elementos nulos en la diagonal. El número de elementos iguales a 1 en la fila i de *A(G)* es $g^+(v_i)$, el grado de salida del vértice v_i, mientras que el número de elementos iguales a 1 en la columna i de *A(G)* es $g^-(v_i)$, el grado de entrada del vértice v_i.

Del digrafo inferior veremos su matriz de adyacencia:

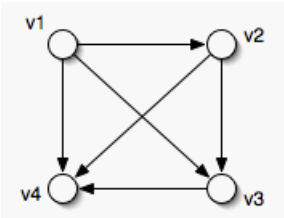

Figura 3.1: Digrafo D1

	v1	v2	v3	v4
v1	0	1	1	1
v2	0	0	1	1
v3	0	0	0	1
v4	0	0	0	0

Cuadro 3.1: Matriz de adyacencias de digrafo D1

3.1.2. Otro tipo de matrices

Otro tipo de matriz diferente al anterior no muy usado.

La matriz se compone de *1, 0 y -1* de manera que:

▷ Cuando es 0 los vértices no tienen ningún arco entre sí.

▷ Cuando es 1 implica que de ese nodo surge un arco hacía el otro.

▷ Cuando es -1 implica que el nodo recibe un arco del otro.

Para mayor facilidad de comprensión vemos un ejemplo:

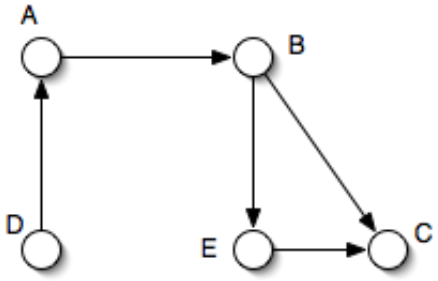

Figura 3.2: Digrafo D2

	A	B	C	D	E
A	0	1	0	-1	0
B	-1	0	1	0	1
C	0	-1	0	0	-1
D	1	0	0	0	0
E	0	-1	1	0	0

Cuadro 3.2: Matriz de digrafo D2

Este estilo de matriz se puede calcular de otra manera: A - A^t
Esto es, restando a la matriz de adyacencia su traspuesta.

Otro tipo de matrices, las denominadas matrices de incidencias, son frecuentemente utilizadas pero de ellas no hablaré en este trabajo.

3.2. Representación mediante listas

3.2.1. Listas de adyacencias

3.2.1.1. Grafo

La lista de adyacencias de un grafo G, con conjunto de vértices $V=\{v_1,v_2,...,v_n\}$, es una lista formada por n sublistas, una para cada vértice v_i, donde figuran los vértices adyacentes al correspondiente vértice v_i.

A continuación pasamos a ver un ejemplo:

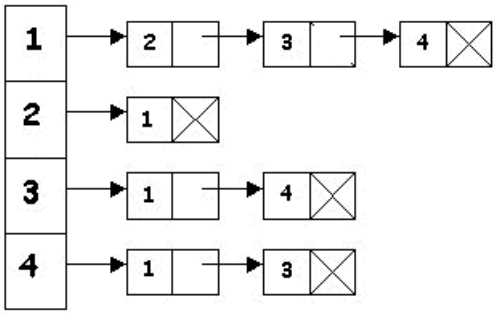

Figura 3.3: Lista de adyacencias de un grafo

Esta lista representa este grafo:

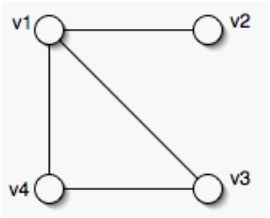

Figura 3.4: Grafo de la lista de adyacencias

Es decir *({2,3,4},{1},{1,4},{1,3})* son sus listas de adyacencias.

3.2.1.2. Digrafo

La lista de adyacencias de un digrafo G, con conjunto de vértices $V=\{v_1,v_2,\ldots,v_n\}$, es una lista formada por n sublistas, una por cada vértice v_i, donde figuran los vértices v_j hacia los que son adyacentes cada uno de los vértices v_i.

Por tanto una lista de adyacencias sería:

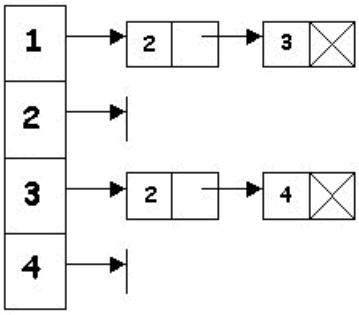

Figura 3.5: Lista de adyacencias de un digrafo

Y su digrafo correspondiente:

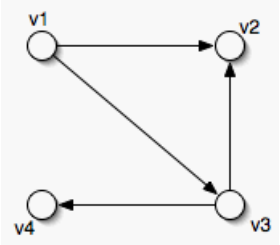

Figura 3.6: Digrafo de la lista de adyacencias

Así que *({2,3},{},{2,4},{})* serían las listas de adyacencias.

El número de elementos de los que está compuesto la sublista i es $g^+(v_i)$, el grado de salida del vértice v_i.

Otra forma alternativa, también utilizando listas entrelazadas, es la representación mediante Listas de Aristas de las que no hablaré en este trabajo.

3.3. Ventajas e inconvenientes de las representaciones

La estructura de los datos utilizada para representar grafos depende de la naturaleza de los datos y de las operaciones que se van a llevar a cabo. Al escoger cuál será la mejor manera de representar nuestros grafos tendremos que tener en cuenta diferentes factores como el número de nodos, número de aristas, si el grafo es o no dirigido, si será necesario agregar o eliminar un nodo o arista y cualquier otro tipo de manipulación a la que se puedan ver llevados.

Entra un nuevo concepto en juego, necesario para determinar la cantidad de datos que tenemos que manejar, **la densidad** del grafo.

3.3.1. Densidad

La **densidad** implica la conectividad de un grafo, esto es un grafo de densidad 0 es un grafo completamente *"desconectado"*, mientras que un grafo de densidad 1 está completamente *"conectado"* (todos los nodos están conectados entre sí con el resto de los nodos).

La densidad de cada grafo variará pues entre 0 y 1 siendo 0 el grafo nulo y 1 el grafo completo.

Podemos construir unas ecuaciones de densidad:

$\frac{2|E|}{n(n-1)} \leq 1 \Rightarrow$ para el caso de grafos, siendo n $\geq$1.

$\frac{|E|}{n(n-1)} \leq 1 \Rightarrow$ para el caso de digrafos, siendo n $\geq$1.

Siendo n el grado del grafo/digrafo y $|E|$ el número de aristas.

Como se puede ver la diferencia para grafos/digrafos es que los digrafos usan arcos y los digrafos aristas.

Poniendo un ejemplo, un grafo que tuviera una densidad de 0.63 estaría mas cerca de ser un grafo completo que uno que tuviera una densidad de 0.25.

3.3.2. Alternativas de representación

La primera alternativa que se nos ocurre para la representación de grafos es mediante una **matriz de adyacencias** cuya mayor ventaja es la facilidad de manipulación y la facilidad de acceso.

Pero esta representación cuenta con bastantes desventajas, la primera es que si el grafo tiene un número pequeño de aristas la matriz tendrá un cuantioso número de ceros, lo que equivale a espacio de memoria desperdiciado.

Otra desventaja es la asignación estática de memoria, por lo que si se desconoce el número de nodos iniciales que utilizará la matriz y después se requieren añadir nodos durante los siguientes procesos es necesario declarar los arreglos con un mayor espacio que el requerido, para poder permitir dichos cambios.

Aparte si se requiere agregar más información sobre los nodos se necesitará declarar una estructura aparte.

Hay un posible arreglo para esta representación que sería guardar exclusivamente la mitad de la matriz ya que es simétrica a partir de su diagonal. Con este arreglo manejariamos la misma información con la mitad de la memoria.

Este tipo de representación es utilizada para grafos muy densos.

Las **listas enlazadas** ofrecen una buena alternativa para representar grafos y cuya ventaja principal es el manejo dinámico de la memoria, lo cual permite de una forma más flexible realizar operaciones con sus nodos del tipo de agregar nodo, eliminar nodo, etc, además de hacer un mejor uso de la memoria que el utilizado por las matrices.

Entre las desventajas que podemos citar en la representación por listas es que debemos utilizar una cantidad de memoria directamente proporcional al número de aristas del grafo para almacenar los apuntadores.

Otra desventaja es que, para los grafos no dirigidos, cada arista apareceria repetida en las listas correspondientes a sus dos extremos.

Este tipo de representación es utilizada para grafos poco densos.

Para el diseño de este proyecto ambas opciones han sido elegidas. Las matrices para la parte de manejo de ficheros y las listas para la parte algorítmica de pura programación.

Parte II

Nauty

Capítulo 4

Herramienta generadora de grafos

4.1. ¿Qué es el Nauty?

Nauty es un programa creado por Brendan McKay *(Ver* [11]*)* compuesto por diferentes herramientas para la ayuda del estudio del grupo de automorfismos de un grafo, de hecho, las siglas "**nauty**" significan "**n**o **aut**omorphism,**y**es?"

Es decir, el programa se ocupa de:

- Isomorfismo entre grafos
- Automorfismos de un grafo

Dentro del programa se encuentran una serie de herramientas facilmente compilables para sistemas linux, son las denominadas **gtools**.

Usaremos dichas funciones para la generación de grafos y digrafos necesarios para realizar más adelante los cálculos de excentricidades. El funcionamiento será explicado en próximos apartados.

La manera de compilar estas herramientas es sencilla teniendo en cuenta que el autor ha adaptado un makefile para su compilación explícita aunque también se puede editar el makefile de manera que sólo compilemos los programas de gtools que nos puedan resultar interesantes.

En nuestro caso inicial no variaremos el makefile aunque de primeras sólo usemos la herramienta generadora de grafos y digrafos denominada **geng.**

La manera de compilar las gtools se hace mediante el comando:

#./make gtools

Dicho esto pasamos a explicar la utilización y código de dicha herramienta.

4.2. Geng, utilización y contenido

A través de la compilación de esta herramienta de código C (como todo el Nauty) podemos buscar la manera de generar los grafos que nos interesen para diferentes casos.

La herramienta tiene diferentes opciones según nuestros intereses, y estas opciones se encuentran detalladas en su comando de ayuda:

```
root@ubuntu:/home/fer/Desktop/TFC/nauty22 # ./geng -help

Usage: geng [-cCmtfbd#D#] [-uygsnh] [-lvq] [-x#X#] n [mine[:maxe]] [res/mod] [fi
le]

 Generate all graphs of a specified class.

      n    : the number of vertices (1..32)
 mine:maxe : a range for the number of edges
              #:0 means '# or more' except in the case 0:0
   res/mod : only generate subset res out of subsets 0..mod-1

     -c    : only write connected graphs
     -C    : only write biconnected graphs
     -t    : only generate triangle-free graphs
     -f    : only generate 4-cycle-free graphs
     -b    : only generate bipartite graphs
                 (-t, -f and -b can be used in any combination)
     -m    : save memory at the expense of time (only makes a
                 difference in the absence of -b, -t, -f and n <= 28).
     -d#   : a lower bound for the minimum degree
     -D#   : a upper bound for the maximum degree
     -v    : display counts by number of edges
     -l    : canonically label output graphs

     -u    : do not output any graphs, just generate and count them
     -g    : use graph6 output (default)
     -s    : use sparse6 output
     -y    : use the obsolete y-format instead of graph6 format
     -h    : for graph6 or sparse6 format, write a header too

     -q    : suppress auxiliary output (except from -v)
```

Figura 4.1: Ayuda geng

Debido a que este trabajo se basa en el cálculo de excentricidades nos basaremos en la primera opción del geng, la opción *-c* para grafos conexos.

Para futuras comprobaciones existen unas tablas que contienen el número de grafos y digrafos conexos diferentes que se conocen con un número de vértices determinado.

(La referencia a estas tablas se puede encontrar en [12])

Connected graphs with n vertices, for n = 1, 2, . . . , 25

n	*connected graphs*
1	1
2	1
3	2
4	6
5	21
6	112
7	853
8	11117
9	2 61080
10	117 16571
11	10067 00565
12	16 40598 30476
13	5033 59078 69219
14	29 00348 74628 48061
15	31397 38114 27612 41960
16	639 69560 11322 51761 76277
17	24 58718 31682 08402 65195 28568
18	1 78733 17252 48899 08889 02005 76580
19	24636 02142 93998 67655 32265 07596 81644
20	6454 65483 19872 27994 26731 12879 45022 83004
21	3221 96273 85046 58981 82320 44119 08215 73234 36797
22	3070 81426 21757 29723 89556 84543 99186 82950 95501 71755
23	5599 43868 82108 80679 65169 46729 81026 45124 90467 68525 69700
24	19570 43463 52067 86942 41449 39394 11275 95390 46378 87524 67290 14803
25	1 31331 19786 44292 67343 03846 15711 21231 90135 75463 49778 70337 47441 10480

Figura 4.2: Tabla de grafos conexos hasta n=25

n	*digraphs*	*connected digraphs*
1	1	1
2	3	2
3	16	13
4	218	199
5	9608	9364
6	15 40944	15 30843
7	8820 33440	8804 71142
8	179 33591 92848	179 24739 55306
9	13 02795 68243 99552	13 02616 16824 66252
10	3 41260 43195 29725 80352	3 41247 40039 94007 65678
11	3 25229 09385 05588 61111 97440	3 25225 68098 54811 53775 95264

Figura 4.3: Tabla de digrafos conexos

El ejemplo más sencillo es la generación de grafos de dos nodos:

```
root@ubuntu:/home/fer/Desktop/TFC/nauty22 # ./geng 2 -c
>A ./geng -cd1D1 n=2 e=1
A_
>Z 1 graphs generated in 0.00 sec
```

Figura 4.4: Ejecución de dos nodos

Sólo existe un grafo conexo con dos nodos y una arista, como podemos comprobar en la tabla de grafos conexos.

La representación del resultado es mediante código ASCII. De esta manera y siguiendo el ejemplo anterior vemos que el grafo resultante es el $A\_$.

¿Pero cuál es la explicación de este resultado? Pasamos a ver la tabla de códigos ASCII.

Letter	ASCII	CodeBinary	Letter	ASCII	CodeBinary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

Figura 4.5: Tabla ASCII 1

Pero el Nauty no sólo usa los códigos ASCII de las letras sino que hay carácteres especiales que también corresponden al resultado y de los cuales también sería útil conocer su código ASCII.

Decimal	ASCII	Binary
32	blank	00100000
33	!	00100001
34	"	00100010
35	#	00100011
36	$	00100100
37	%	00100101
38	&	00100110
40	(	00101000
41	)	00101001
42	*	00101010
44	,	00101100
45	-	00101101
46	.	00101110
91	[	01011011
92	/	01011100
93	]	01011101
94	^	01011110
95	_	01011111
96	`	01100000
123	{	01111011
124	|	01111100
125	}	01111101
126	~	01111110

Figura 4.6: Tabla ASCII 2

Para más facilidades hay conversores por internet que hacen la traslación ASCII-Binario de manera automática:

htttp://www.theskull.com/javascript/ascii-binary.html

Nuestro resultado para el grafo conexo de dos nodos era $A\_$ que pasado a binario es *0100000101011111*.

Este código es un tipo de formato de datos específico para la codificación de grafos que pasaremos a aclarar a continuación.

4.3. Formato de datos de codificación

4.3.1. Graph6 (g6)

Hay dos valores que toman interés en esta representación:

- ***N(n)***, la codificación del orden n del grafo.
- ***R(x)***, la codificación de las aristas de los vértices.

Supongamos un grafo de orden 5 cuyas aristas fueran:
((0,2), (0,4), (1,3), (3,4))

4.3.1.1. N(n)

Seguimos con el ejemplo anterior, nuestro grafo de orden 5.

Para el cálculo del formato en *bigendian* debemos sumar **63** al orden del grafo, por lo que N(n) = 68.

Le sumamos 63 para que todas las computadoras tengan la misma codificación de formato debido a la importancia de los bits del final.

4.3.1.2. R(x)

Realizamos una codificación inicial mediante la forma:
(0,1), (0,2), (1,2), (0,3), (1,3), (2,3), ... (n-1,n)

Este estilo de codificación es muy útil debido a que si se añaden nuevos vértices con sus nuevas aristas estos se situarán al final de la codificación y no tendremos que variar el orden de las codificaciones anteriores.

En orden de los enlaces los que se encuentran en nuestro grafo los representamos mediante un 1 y los que no se encuentran mediante un 0.

Así la **x** final será → x = 0100101001

Ahora debemos realizar la R(x).

Dividimos la x en grupos de 6 elementos desde el inicio (izquierda) de la x y completamos con 0 si la x no es múltiple de 6 como es nuestro caso de ejemplo.

R(x) = R (010010 100100) = 81 99

4.3.1.3. Solución del grafo ejemplo

En nuestro caso nuestro grafo ejemplo sería *68 81 99* siguiendo el formato graph6.

4.3.2. Sparse6 (s6)

Este formato de datos tiene una extensión específica (*.s6*) para diferenciarla de los demás formatos.

Cada grafo ocupa exclusivamente una única línea de texto y exceptuando por el carácter final de línea cada carácter tiene la forma $63 + x$, donde $0 \leq x \leq 63$. El byte codifica 6 bits de x.

La codificación consiste en:

- Carácter *: (distingue así de los otros formatos)*
- Número de vértices (orden)
- Lista de aristas
- Final de línea *(end of line)*

Usaremos un ejemplo para poder retratar muchisimo mejor el formato:

:Fa@x^

¿Este código representa a un grafo? Veamos como podemos asociar este texto a un grafo.

(**:**) indica que el grafo está en formato **s6**.

Las demás letras/símbolos las codificamos en binario y les restamos 63 de nuevo para salvar el problema del *bigendian.*

Nos queda así: 000111 100010 000001 111001 011111

El primer byte no es mayor de 63 así que se trata sin duda a *n*. Por lo que n = 7 (disponemos de 7 vértices en nuestro grafo).

Para codificar las aristas necesitamos los bits suficientes para codificar n-1.

En nuestro caso, 7 - 1 = 6 $\Rightarrow$ 3 bits $\Rightarrow$ K = 3

4.4. Conclusiones de NAUTY

Esta herramienta ha sido realmente útil para la finalización del proyecto debido a la numeración de todos los grafos conexos de un orden dado no isomorfos.

Mediante los *.txt* generados con el programa de Brendan McKay hemos podido crear una aplicación JAVA que utilice sus parámetros para sacar sus propias conclusiones.

Es por ello, que hemos incluido esos documentos ya generados dentro de la misma herramienta para que nos sea más fácil poder acceder a dichos datos en cualquier momento sin tener que compilar la propia herramienta en C.

Si se quisiera ampliar la utilización de esta, sólo se tendrían que incluir los nuevos ficheros generados con NAUTY.

Parte III

La herramienta

Capítulo 5

Implementacion de la herramienta

En este capitulo explicaremos la elección del lenguaje de programación asi como las razones que nos han llevado a utilizarlo, además de una pequeña definición de las clases que conforman la herramienta.

5.1. Lenguaje de desarrollo

Para la implementación de esta aplicación se ha de emplear un **lenguaje orientado a objetos** como es el JAVA. Inicialmente se había previsto la realización en C++ pero se desestimó debido a las numerosas librerías de JAVA que pueden facilitar la gestión de numerosas operaciones.

Las razones por las que se ha elegido este lenguaje:

1. Su amplia difusión.
2. Su capacidad para reutilizar el código.
3. Su capacidad de abstracción.
4. El hecho de que se trata de un lenguaje orientado a objetos.
5. Su portabilidad entre diferentes plataformas.

La **abstracción** es la base de la programación orientada a objetos y se basa en identificar los servicios que nos ofrece y no la forma en la que están implementados.

Otro término importante es la **encapsulación**, con la cuál podemos separar y diferenciar dos claros niveles como son la *interficie* (que indicará los servicios ofertados) y la *implementación* (la forma en la que estos servicios están desarrollados)

5.2. Desarrollo de la aplicación

Para el implementación de esta librería se ha tenido que desarrollar una serie de clases y ficheros auxiliares. Algunas clases de éstas han sido desarrolladas en la asignatura de **EDI** durante la carrera como veremos a continuación. De todas maneras han sido trasladadas a JAVA que fue finalmente el sistema empleado.

5.2.1. GEM.exe

El ejecutable de la aplicación. Necesario para acceder al menú inicial que accede a todas las clases realizadas.

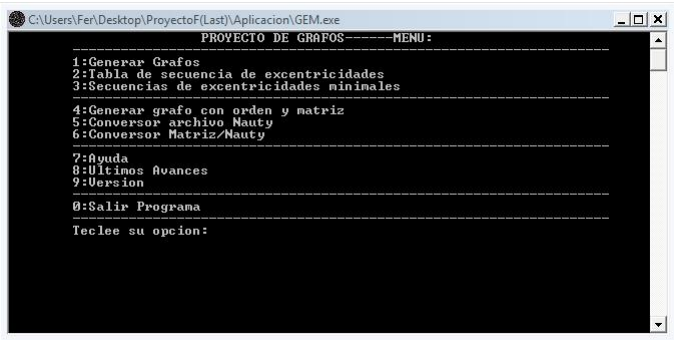

Figura 5.1: Menu GEM.exe

5.2.2. Clases programadas

Dentro de nuestra librería encontramos numerosas clases programadas para programación interna y acceso a los objetos Graphs y GraphsTable.

5.2.2.1. Clase lista

Mediante esta clase definimos las operaciones para las estructuras de datos conocidas como listas. Se trata de una clase genérica que trata cualquier elemento que podría estar en una lista.

Es una de las clases definidas durante la carrera en la asignatura de **EDI** y su código es facil de adaptar a nuestro trabajo. Sus funciones no representan mucha dificultad de comprensión por lo que no ahondaremos más en esta clase.

5.2.2.2. Clase cola

Muy parecida a la clase lista, esta clase también es genérica. Y, al igual que la clase lista, se trata de una subclase de la clase ed, heredando pues sus propiedades.

La clase cola se utiliza para el trabajo auxiliar con los digrafos por lo que nos encontramos con una clase secundaria dentro de nuestra aplicación.

5.2.2.3. GraphFactory

La factoría de grafos es el centro de nuestra aplicación con numerosas funciones que generan grafos aleatorios (otro sistema, diferente al de Nauty), genera matrices y calcula excentricidades.

5.2.2.4. ConvertNauty

Esta clase es una herramienta útil para la conversión de los sistemas de codificación de Brendan a mi propio sistema de codificación.

Nauty emplea (como hemos visto en capitulos anteriores) la codificación Graph6. Mediante esta clase podemos convertir esta codificación en su grafo, y más adelante en su matriz de adyacencias para poder ser guardado en el *txt*.

5.2.3. Objetos definidos para nuestra aplicación

1 **UndirectedSimpleGraph**: objeto grafo.

2 **Node**: objeto vértice para grafos.

3 **Edge**: objeto arista para grafos.

4 **GraphTable**: objeto para generar la tabla de excentricidades.

5 **GraphFactory**: el objeto más interesante para nuestra aplicación, útil para acceder a las funciones de nuestra aplicación.

5.3. Los problemas surgidos durante el desarrollo

5.3.1. El problema de la secuencia de 0's

Durante la implementación de esta aplicación surgió una situación interesante: Al problema de la transformación a código Nauty de las matrices generadas se ideó una manera de realizarla automáticamente. Se trataba de agregar un número de 0's de tal manera que la secuencia fuera múltiplo de 6 para poder pasar a código Nauty.

Se ha realizado un estudio para ver qué números de 0's se debe agregar en cada caso.

Nodos	Número de bits	Total	Múltiplo	Secuencia	SQ
2	2+1	3	6	000	s1
3	3+2+1	6	6	-	-
4	4+3+2+1	10	12	00	s2
5	5+4+3+2+1	15	18	000	s1
6	6+5+4+3+2+1	21	24	000	s1
7	7+6+5+4+3+2+1	28	30	00	s2
8	8+7+6+5+4+3+2+1	36	36	-	-
9	9+8+7+6+5+4+3+2+1	45	48	000	s1

Cuadro 5.1: Tabla del Problema de la Secuencia de 0's

Campos:

Nodos: *El número de nodos del grafo.*

Total: *La suma del número de bits que ocupa cada grafo.*

Múltiplo: *El múltiplo de 6 más cercano del Total.*

Secuencia: *La secuencia de 0's a ingresar.*

Como se puede observar se producen exclusivamente tres casos diferentes para órdenes de 2 a 9.

▷ **Caso 1:** Se introduce una secuencia de tres 0's (s1).

▷ **Caso 2:** Se introduce una secuencia de dos 0's (s2).

▷ **Caso 3:** No se introduce ninguna secuencia.

(Nota: Esta por ver si para órdenes superiores existe una secuencia lógica en la introducción de 0's de tal manera que podamos determinar automáticamente el número a ingresar. Podría resultar interesante realizar un estudio sobre este tema en un futuro.)

5.3.2. El problema de la generación de Grafos

Durante el desarrollo de la aplicación se previó desentrañar la herramienta Nauty para poder realizar la generación de grafos no isomorfos de un orden dado.

Debido a la dificultad de dicha herramienta se intentó realizar una nueva que pudiera hacer la misma generación pero con nuestro código y matrices. En el menú podemos observar dicha resolución *(apartado 1 en nuestra herramienta, poner un 0 para calcular su número)*, tristemente incompleta debido a la dificultad de desentrañar el problema NP del isomorfismo.

La idea inicial era calcular el tamaño de la secuencia de 0's y 1's de la matriz que representaba nuestros grafos e ir sumando en binario de 1 a 1 para evitar su repetición. Con esto solucionabamos el problema de recorrer todos los grafos posibles (ya que todas sus matrices eran, de hecho, diferentes), y tras una medición de los grafos conexos, observábamos que aún nos incluía los isomorfos.

Se tuvo que abandonar esta idea y dedicarse plenamente a la utilización de los archivos Nauty.

5.3.3. El problema del tiempo y el espacio

Los ordenadores de hoy en día y su memoría interna están muy avanzados pero el autor calcula que hasta algunos años más no se podrán hacer calculos con dicha aplicación en su plenitud. La generación de grafos de orden 9 es, por ahora, un cómputo casi imposible debido a la necesidad de una mayor cantidad de memoría interna que necesita el programa.

Aparte, su generación en cuanto al tiempo aumenta de manera exponencial y podría resultar un cómputo de varias horas, aunque se ejecutara en paralelo.

El tiempo debería solucionar este problema.

5.4. Diseño de interacción

5.4.1. Generación de la tabla de excentricidades

Entrada	Fichero con grafos en formato de matriz
Salida	Tabla de excentricidad por fichero

Cuadro 5.2: Especificación TablaEx

Mediante la entrada de un documento de texto *(.txt)* calcula una tabla de excentricidad.

Secuencia	**Notación abreviada**	**Frecuencia**	**Medida**
1,1	1^2	1	1

Cuadro 5.3: Ejemplo Tabla Excentricidad para grafos de orden 2

En nuestro modelo de salida de datos la notación abreviada sigue la pauta: $X^Y \Rightarrow$X(Y).

5.4.2. Generación de grafos aleatorios

Entrada	Orden y número de grafos por teclado
Salida	Por pantalla/fichero, en formato de matriz

Cuadro 5.4: Especificación TablaEx

Se generan unos grafos siguiendo las opciones por pantalla, con la posibilidad de crear un .txt para guardar los datos.

5.4.3. Conversión Nauty

Entrada	Fichero con grafos en Nauty
Salida	Fichero convertido de matrices

Cuadro 5.5: Especificación TablaEx

Se cogen los grafos en Nauty de un fichero y se convierten a un nuevo fichero de datos, guardando sus respectivas matrices.

Parte IV

Conclusiones y trabajos futuros

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo veremos cuales son las conclusiones a las que hemos llegado después de realizar el proyecto asi como las futuras mejoras o trabajos a realizar a partir de nuestra aplicación.

6.1. Conclusiones

Durante el transcurso de la realización se han tenido que solucionar numerosos problemas. El primero de ellos se trataba de desentrañar la adaptación del sistema Nauty pero debido a su alta complejidad ha sido descartado. Durante un tiempo se buscó un nuevo sistema de generación, incluido en esta aplicación. Se generaban todas las matrices posibles para un orden, tras lo cuál, se miraban si reflejaban grafos conexos. Pero de nuevo nos encontrabamos con las dificultades del isomorfismo (problema NP).

Finalmente se solucionó obteniendo los ficheros de grafos de Nauty y generando las tablas de excentricidades a partir de ellos.

El autor espera que dicha aplicación sirva para acelerar la resolución del problema de la excentricidad y avanzar en la matería de la teoría de grafos.

6.2. Posibles trabajos futuros

Numerosos frentes han quedado abiertos:

1 Toda la aplicación podría ser reversionada para la inclusión de digrafos.

2 La medición de secuencias de excentricidades minimales *(Visto en el apartado 2.3.2)*

3 El paralelismo entre el modelo de programación para acelerar la ejecución de trozos de código en diferentes computadores.

4 El problema de la secuencia de 0's: para órdenes superiores determinar la existencia de una secuencia lógica de inclusión de 0's. *(Visto en el apartado 5.2.5.2)*

6.3. Programas, aplicaciones y links de interés

6.3.1. Links

▷ ***htttp://www.theskull.com/javascript/ascii-binary.html*** ⇒ Conversión de ASCII a Binario de manera automática.

6.3.2. Aplicaciones y programas

▷ ***OmniGraffle (Mac OS X)*** ⇒Herramienta de dibujo de grafos.
▷ ***LyX*** ⇒Herramienta de texto para LateX.
▷ ***gcc y g++*** ⇒Compiladores para lenguaje C y C++ respectivamente.
▷ ***Eclipse***⇒Herramienta para compilar en JAVA.

Bibliografía

[1] M. Behzad, J. E. Simpson, Eccentric sequences and eccentric sets in graphs, *Discrete Math,* **16,** (3), 1976, 187-193.

[2] G. Chartrand, Distance in Stratified Graphs, *Czechoslovak Mathematical Journal,* **50(125),** (1), 2000, 35-46.

[3] G. Chartrand, O. R. Oellermann, *Applied and Algorithmic Graph Theory,* McGraw-Hill, 1993.

[4] M. Downes, *Short Math Guide for Látex,* American Mathematical Society, 2002.

[5] J. Gimbert, N. López, Eccentricity sequences and eccentricity sets in digraphs, *Ars Combinatoria (aceptado).*

[6] R. Grimaldi, *Matemática Discreta y Combinatoria,* Addison-Wesley Iberoamericana, 1997.

[7] M. Gordon, M. Minoux, *Graphs and Algorithms,* Wiley-Intercience, 1984.

[8] L. Joyanes, *Turbo C++: Iniciación y Referencia,* Serie McGraw-Hill de Informática, 1997.

[9] D. Jungnickel, *Graphs, Networks and Algorithms,* Springer Verlag, 1999.

[10] L. Lesniak, Eccentric Sequences in Graphs, *Periodica Mathematica Hungarica,* **6,** (4), 1975, 287-293.

[11] B. D. McKay, *Nauty User's Guide,* Computer Science Department of Australian National University, 2002.

[12] R. C. Read, R. J. Wilson, *An Atlas of Graphs,* Clarendon Press, 1998.

[13] J. J. Salazar, *Lecciones de Optimización,* Universidad de La Laguna, 2000.

[14] J. A. Sánchez, J. C. Valverde Fajardo, *Algebra y Matemática Discreta,* Popular libros, 2000.

[15] B. Pelegrin, L. Cánovas, *Algoritmos en Grafos y Redes,* PPU, 1992.

Parte V

Anexo